UNIVERSITAT DE BARCELONA

Trabajo final de grado

GRADO DE INGENIERíA INFORMÁTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

Clasificación automática de mensajes de un marketplace en función de su contenido

---

Autor: Eudald Arranz Tresserra

Directores: Eloi Puertas

Realizado en: Departament de Matemàtiques i Informàtica

Barcelona, 27 de junio de 2019

Abstract

This project discloses a solution to a problem that many transactional marketplaces are facing: How to stop users from using the message system of the platform to close the deal with the service provider without paying the fees involved in the transation. *Holidog*, a marketplace that provides services related to pets, needs to solve this challenge using an autonomous approach. In these pages, you will find the analysis and solution designed for them: A filter based on the content of the messages to filter away the ones that go against the terms and conditions of the platform, with good enough results for production use.

Resumen

Este proyecto busca solucionar un problema común en los marketplaces transaccionales: Evitar que los usuarios usen el sistema de mensajería de la plataforma para cerrar un acuerdo con los proveedores de los servicios y no pagar las comisiones relacionadas con la transacción. *Holidog*, un marketplace que proporciona servicios relacionados con el cuidado de las mascotas, necesita una solución automática para este problema. En este proyecto encontrarás el análisis y la solución que se ha diseñado para ellos: Un filtro basado en el contenido de los mensajes para filtrar aquellos que puedan ir en contra de las condiciones de uso de la plataforma, con unos resultados lo suficientemente buenos como para poder usarse en producción.

Agradecimientos

Este trabajo ha sido posible gracias a Facundo Farias, responsable tecnológico de *Holidog*, que confió en mis capacidades para la realización de este proyecto y a Eloi Puertas, que lo ha guiado y tutorizado.

A Clara y a Ot por su paciencia y cariño.

Índice

1. Introducción

1.1. Ámbito del proyecto

El mundo de la gestión y procesamiento de datos, así como de la interpretación y análisis de los mismos, ha sufrido una revolución en la última década. La cantidad de datos que como sociedad somos capaces de guardar y procesar no tiene precedentes.

Esto, junto a la aparición de nuevas tecnologías y técnicas de procesado y a la bajada de los costes de la capacidad de procesamiento, abre la posibilidad a la implementación de soluciones de procesado de datos masivos a agentes del mercado que antes no podían acceder a ellas.

A partir del 2006, y con la entrada de grandes actores como *Google*, *Amazon*, *IBM* o *Microsoft*, la evolución del aprendizaje automático ha permitido solucionar problemas que hasta ahora resultaban imposibles de solucionar a nivel comercial. En especial, problemas relacionados con la visión artificial, el procesamiento del lenguaje natural, o la clasificación y etiquetado de datos tienen ahora soluciones comerciales con resultados excelentes.

En este ámbito es donde nace este proyecto. *Holidog*, empresa dedicada a ofrecer servicios para mascotas a través de particulares ha ido guardando una gran cantidad de datos relacionados con su actividad durante los últimos años. Con estos datos se abre la posibilidad de automatizar procesos de su modelo de negocio.

Uno de estas tareas que puede ser automatizada es la moderación de los mensajes que, a día de hoy, recae en los trabajadores de la empresa. *Holidog* permite a sus usuarios enviarse mensajes relativos a los servicios que sus clientes ofrecen o solicitan. El propósito de este proyecto es automatizar la moderación de estos mensajes a partir del histórico de mensajes guardados.

1.2. Contexto

La clasificación de mensajes es un problema canónico del Procesamiento del lenguaje natural (de ahora en adelante PLN), el campo de las ciencias de la computación que se encarga de trabajar y estudiar el lenguaje natural y su procesamiento.

Desde los inicios de la disciplina la clasificación de texto ha sido aplicada en problemas prácticos. El interés en estas aplicaciones ha sido uno de los motores en el crecimiento y desarrollo del PLN. Otros problemas habituales son la traducción automática, la predicción de texto, la comprensión del lenguaje (tanto escrito como oral), la generación de texto o la síntesis del mismo.

La mayoría de estos problemas han empezado a tener aplicaciones prácticas en los últimos años, con la aparición de herramientas como los asistentes virtuales, los teclados inteligentes o la nueva generación de traductores automáticos.

En el caso concreto de la clasificación de texto, un claro ejemplo son los filtros de SPAM. Históricamente, estos eran filtros de contexto, es decir, se determinaba si un correo era SPAM o HAM en función de variables externas al mensaje: su origen,

el historial de la persona que lo enviaba, la hora, el número de mensajes enviados simultáneamente...

Sin embargo, aplicando solo estos filtros, algunos email SPAM pueden llegar a ser considerados como buenos. Es aquí donde la función del análisis de contenido gana importancia. Si los filtros son capaces de entender el contenido de un determinado mensaje (ver mensaje 1) este será considerado SPAM independientemente de su contexto.

> Good day!
> We would like to offer cheapest Viagra in the world! You can get it at:
> LINK
> Sincerely,
> Liza Stokes

Mensaje 1: Email SPAM

Puede ser considerado SPAM independientemente de su contexto, si somos capaces de entender el contenido del mensaje podemos determinar su intencionalidad.

A medida que los métodos de procesamiento del lenguaje mejoraban, el foco del filtro se ha ido moviendo del contexto al contenido. Métodos simples como la búsqueda de palabras clave o métodos mucho más sofisticados que buscan entender la semántica de los mensajes han ido mejorando el rendimiento de los filtros. Con este enfoque Google afirmó en verano de 2017 que eran capaces de filtrar un 99,9 % de los emails correctamente [14].

1.3. Motivación

En *Holidog* son conocedores de la ventaja competitiva que supone tratar y trabajar con los datos recopilados por la plataforma. Este proyecto nace de la motivación de automatizar algunas de las tareas que realiza la empresa usando los datos recogidos.

El objetivo de la plataforma es claro: poner en contacto a cuidadores de mascotas con los propietarios que precisen de sus servicios. Estos usuarios pueden explorar el catálogo de cuidadores, ver sus fichas y seleccionar aquellos que sean más afines a sus necesidades.

Para poder encontrar al cuidador ideal, *Holidog* permite el intercambio de mensajes entre usuarios de manera gratuita. Una vez seleccionado el cuidador deseado, el usuario final necesita pagar una subscripción para poder acceder a los datos de contacto del cuidador y así poder formalizar la reserva.

Esta fase previa en la que los usuarios pueden enviarse mensajes es peligrosa a nivel de negocio, ya que es muy fácil que los usuarios envíen su información personal a través de los mensajes, evitando así pagar la cuota de subscripción. El modelo

de negocio se basa en estas subscripciones así que es vital que estos mensajes sean moderados para evitar la transmisión de contenido sensible.

Hasta ahora, la moderación de estos mensajes se hace de manera manual, sin ningún tipo de automatización. Una serie de trabajadores autónomos se encargan de comprobar el contenido de los mensajes uno a uno y moderarlos. Este método no es solo repetitivo y lento sino que además supone un coste extra para la empresa.

Durante todo este tiempo se ha ido guardando el resultado de esta moderación en las bases de datos de la plataforma. La dirección técnica de *Holidog* plantea usar esta información para la automatización parcial o total del filtrado de los mensajes.

Este proceso podría parecer un problema simple de procesamiento del lenguaje natural. Sin embargo, como se expone en las páginas siguientes, los usuarios usan todo tipo de códigos para ocultar información e intentar engañar así a los moderadores de los mensajes. Este proyecto presenta algunas soluciones para tratar de generalizar la detección de estos códigos basándose en conceptos de Esteganografía. [1]

1.4. Objetivos generales

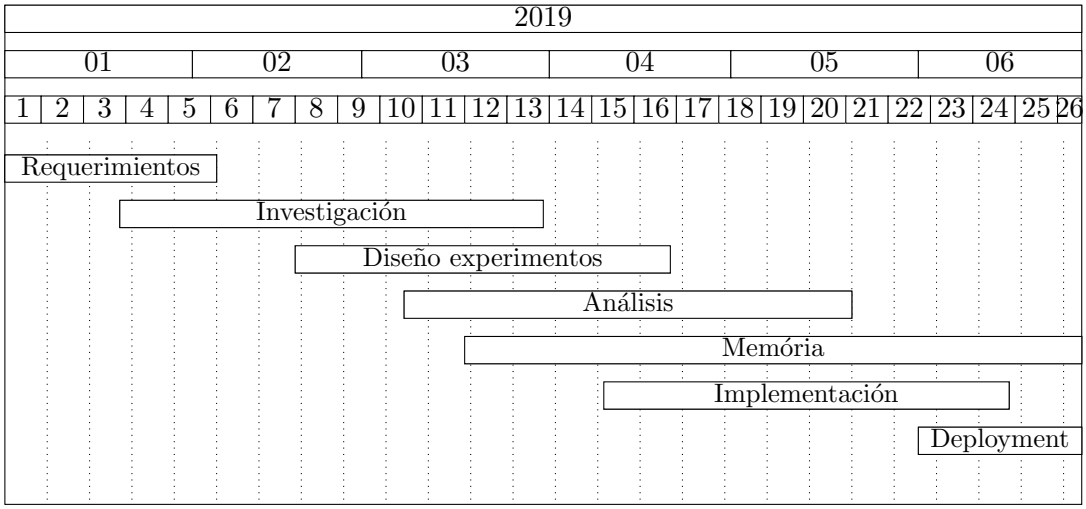
El objetivo de este proyecto es la creación de un sistema de clasificación automático de estos mensajes usando la extensa base de datos de la empresa *Holidog*.

1.5. Objectivos específicos

Los objetivos específicos del proyecto son los siguientes:

- Investigación de las técnicas de clasificación de texto.
- Investigación de las técnicas del procesamiento del lenguaje natural.
- Analizar y entender los datos de los que *Holidog* dispone.
- Diseñar un modelo que permita predecir la clasificación de los mensajes y una implementación para poder integrar la solución.
- Implementar el modelo y la solución.

1.6. Planificación temporal



1.7. Organización de la memoria

Este proyecto seguirá la siguiente estructura:

- Estudio del estado del arte: la clasificación de texto por contenido semántico no es un problema resuelto, es importante entender qué técnicas se usan y la historia de este campo.
- Análisis del problema: para poder proceder a implementar una solución se requiere entender muy bien el problema. Se analizarán los datos disponibles y los requerimientos de *Holidog*
- Diseño de las soluciones: se diseñarán varios experimentos para encontrar el mejor modelo para el problema. Además se diseñará una solución software para poder integrar los modelos a la plataforma.
- Resultados y implementación: se analizarán los resultados de los experimentos y la implementación de la solución.
- Conclusiones y trabajo futuro: conclusión del trabajo, revisión de los objetivos logrados y el trabajo pendiente.

2. Estado del arte

Este proyecto presenta un problema típico de clasificación de texto: partiendo un un conjunto de mensajes etiquetados con una de las tres clases posibles (mensaje aceptado, mensaje corregido o mensaje prohibido) se neceista crear un modelo capaz de asignar a un mensaje nuevo la clase que le pertenece.

Para entender este tipo de problemas es necesario analizar la historia de el procesamiento del lenguaje natural (PLN) y las técnicas que se usan actualemente para intentar resolver este tipo de problemas.

El procesamiento del lenguaje natural es una rama mixta de la lingüística y las ciencias de la computación que se encarga del estudio y procesamiento del lenguaje natural, es decir, , tanto escrito como hablado, que constituye los miles de idiomas que usa el ser humano para comunicarse.

Aunque parezca un campo emergente y poco conocido; sus avances nos rodean en nuestro día a día. Hace no muchos años el uso de una interfaz de usuario por voz parecía ciencia ficción. Ahora, dispositivos como los asistentes virtuales (*Amazon echo, Google Home, Siri de Apple o Cortana*) usan interfaces por voz realmente eficientes [23].

Detrás de estos dispositivos hay aplicadas todo tipo de soluciones que provienen directamente del PLN. Estos dispositivos necesitan ser capaces de entender aquello que reciben transformando las ondas en palabras o tokens y así poder analizar su significado. Es decir, el proceso se divide en dos fases: la primera busca tokenizar la voz y la segunda entender aquello que el usuario quiere decir. Además, estos dispositivos también usan avances en la síntesis de voz, puesto que su respuesta se da de manera oral. Así, no solo se necesita entender el mensaje original y dar con la respuesta a su petición, sino construir un mensaje con sentido y responder con un mensaje sintetizado de voz.

El problema que este estudio trata de resolver es mucho más simple, aún así, usaremos los mismos principios y técnicas para intentar resolverlo.

2.1. Breve historia del procesamiento del lenguaje natural

La traducción automática se puede considerar el origen del PLN. Aunque parezca un campo de estudio muy nuevo, en el siglo XIV Ramón Llull ya teorizó sobre la manera de entender el mundo a través del lenguaje formal, más tarde, figuras clave del Racionalismo filosófico como Descartes o Leibniz empezaron a trabajar sobre modelos teóricos de traducción. Se basaban en la idea de lenguajes universales y lógicos [9]. Las primeras aplicaciones de estas ideas no tardaron en aparecer, como uno de los primeros corpus etiquetados del lenguaje de Johann Becher de 1661. [28]. Estos avances, sin técnicas de computación detrás, quedaron en intentos teóricos de entender la lógica detrás del lenguaje.

Después de la Segunda Guerra Mundial, siguiendo el legado de Alan Turing y sus avances en computación , las técnicas de traducción automática se acercaron al sistema criptográfico que él ideó. Este acercamiento entiende los diferentes idiomas como códigos criptográficos del lenguaje universal. Así, una traducción es entendida como un cambio de código. [28] Estos métodos fueron populares durante la guerra fría, cuando se crearon aplicaciones para la traducción automática entre el ruso y el inglés. El conocido experimente de Georgetown en 1954 generó por primera vez un gran interés mediático alrededor de la traducción automática. [10]

En 1957 Noam Chomsky publicó su libro *Estructuras sintácticas*, donde presentaba un sistema universal para entender los lenguajes. Analizando las estructuras sintácticas de varios idiomas llegó a la conclusión que existe una estructura mental innata que permite la producción y comprensión de un enunciado en cualquier idioma,. Esta teoría se conoce con el nombre de *gramática generativa.* [2]

Este cambio revolucionó la lingüística y los métodos de procesamiento del lenguaje natural, que empezaron a trabajar sobre esta gramática universal, apartándolos de los métodos criptográficos.

La siguiente revolución tuvo lugar en los años 80 y 90 con la consolidación de los métodos estadísticos usados en el *Machine Learning.* Hasta entonces el PLN usaba modelos gramaticales pequeños y escritos a mano, limitando mucho su escalabilidad. A partir de entonces, gracias a la mayor capacidad computacional y a la publicación y creación de *corpus* cada vez más completos y robustos, se crearon por primera vez modelos mucho más grandes generados con estos métodos estadísticos.

En el año 1999 Christopher D. Manning y Hinrich Schütze publican *Foundations of statistical natural language processing* considerado por muchos el manual básico del procesamiento estadístico del lenguaje. [13] Este manual sintetiza las bases de las técnicas y los modelos matemáticos que se usan hoy en día.

2.2. Métodos estadísticos y conceptos

Los métodos estadísticos que usa el PNL basan en un concepto relativamente simple: La tokenización del texto. Este proceso consiste en transformar una cadena de caracteres en una representación matricial de su contenido de esta cadena (palabras, n-grams [1], frases, párrafos, relaciones entre palabras...). Una vez se tiene el texto tokenizado se pueden aplicar métodos de clasificación estadística sobre las matrices.

$$\text{Texto lenguaje natural} \rightarrow \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}$$

En los últimos años se han popularizado varios de estos tipos de tokenizadores, desde el conceptualmente simple Tf-idf al recién publicado BERT que proponen diferentes maneras de realizar esta conversión.

2.2.1. Tf-idf

El Tf-idf o *Term Frequency Inverse Document Frequency* es el método más sencillo de los que se presentan.

[1]Subcojunto ordenado de un conjunto ordenado, en procesamiento del lenguaje natural normalmente hace referencia a una secuencia de n palabras dentro de una cadena de palabras.

Como su nombre indica, consiste en evaluar la inversa de la proporción de la frecuencia de una palabra de un documento (conjunto de palabras) en relación al porcentaje de documentos en la que aparece. [25]

La idea detrás es que las palabras más frecuentes definen menos un documento que aquellas más raras, que tienen una relación más fuerte con el sentido del documento.

El concepto matemático en el que se basa también es relativamente simple, Juan Ramos [25] lo define así:

Dada una colección de documentos D, una palabra w y un documento concreto $d \epsilon D$, podemos calcular su *tf-idf* así:

$$w_d = f_{w,d} \cdot log(\frac{|D|}{f_{w,D}})$$

donde $f_{w,d}$ equivale a la frecuencia de la palabra w en el documento d, $|D|$ al tamaño del corpus y $f_{w,D}$ la frecuencia de la palabra w en el corpus entero de documentos (D).

La simplicidad conceptual de este método lo hace muy fácil de implementar. Primero se define un tamaño de vector n. Después se busca el coeficiente *tf-idf* para cada palabra del corpus. Y finalmente se seleccionan las n más significativas (aquellas n que definen mejor cada documento individual dentro del corpus). Para aquellos documentos fuera del corpus generamos un vector de tamaño n buscando el *tf-idf* de las n palabras más significativas ordenadas.

Este método nos permite tokenizar de manera fácil documentos sobre los que aplicar técnicas estadísticas. Un acercamiento conceptualmente sencillo es aplicar KNN sobre el vector v del nuevo documento y los vectores de cada documento del corpus D. Consiguiendo así los documentos más parecidos a v. [29]

2.2.2. Word2Vec

En septiembre de 2013 Tomas Mikolov y su equipo de *Google* publicaron el estudio *Efficient Estimation of Word Representations in Vector Space*. [15] Este proponía una nueva manera de codificar texto en un espacio vectorial de manera mucho más eficiente. La implementación de este modelo dio lugar a los *Word2Vec*.

Estos modelos propuestos no solo transformaban las representaciones de palabras en vectores, sino que aseguraban un espacio vectorial coherente. Por ejemplo, las representaciones vectoriales de palabras que representan colores están cerca en el espacio vectorial entre ellas. Además, las operaciones entre vectores conservan la coherencia, es decir, la suma de $vec('king')$ con $vec('woman')$ produce un vector que está cerca de $vec('queen')$.

Esto es un cambio significativo, con *tf-idf* conseguíamos saber el sentido general del documento usando sus palabras más representativas, con *word2vec* conseguimos retener la información semántica del documento en su representación vectorial.

El concepto matemático de este método es un poco más complejo. Mikolov propone dos implementaciones diferentes: el Continuous BoW [2] y el Skip-Gram. Ambos tratan de modelar lo siguiente: dado un corpus D queremos encontrar aquellas representaciones que son útiles para predecir las palabras que suelen rodearla. En el estudio *Distributed Representations of Words and Phrases and their Compositionality* lo definen formalmente así: [16]

Dada una secuencia de palabras $w_1, w_2, w_3...w_N$ el objetivo del modelo es maximizar la probabilidad media logarítmica:

$$\frac{1}{N}\sum_{n=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log\, p(w_{n+j}|w_n)$$

La aplicación de estos modelos supuso un incremento de la precisión de varios puntos porcentuales en la mayoría de los benchmarks tipo para validar modelos de PLN.

2.2.3. GloVe

Los *GloVe* (Global vectors for word representation), al igual que los *Word2Vec* intentan codificar representaciones de palabras en forma de vectores en un espacio vectorial coherente, que mantenga la relación semántica y sintáctica.

La idea fue publicada en 2014 por Jeffery Pennington, Richard Socher y Christopher Manning en *Glove: Global vectors for word representation* [21]. Este estudio prometia mejorar la eficiencia de los modelos existentes hasta entonces. A diferencia de los métodos como *Word2Vec* que trabajan con ventanas de contexto, este tiene la capacidad de aprender desde una prespectiva global.

Los autores presentan presentan un modelo que parte de las ideas detrás de los Skip-grams propuestos por Mikolov, pero que en lugar de usar la co-ocurrencia de las probabilidades, usan un ratio de estas probabilidades.

La función de pesos que presenta este método es mucho más complicada que la anterior, quedando fuera del alcance este proyecto su justificación: [21]

$$\hat{J} = \sum_{i,j} f(X_{ij})(w_i^T\tilde{w}_j - logX_{ij})^2$$

La aplicación de este método supera en varios puntos el rendimiento de los métodos existentes hasta ahora; Los resultados destacan especialmente en los modelos de reconocimiento de entidades nombradas (NER).

[2] En PLN, BoW hace referencia a un conjunto de palabras significativas, liternalmente: *Bag of words*.

2.2.4. Transformers y los nuevos métodos

Los últimos avances en modelos de procesado han sido los transformers y los modelos de preprocesado. Tres grandes avances se han producido durante el 2018 que han empujado este desarrollo. Se trata de la publicación de los siguientes tres estudios: *Deep contextualized word representations* [22] (conocidas como ELMo), Improving *Language Understanding by Generative Pre-Training* [24] (conocidos como Generative Pre-trained Transformers) y *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* [4] (conocidos como BERT).

Todos ellos se basan en la idea que empezaron a trabajar los modelos de transformación vectorial: Lo más importante para entender el funcionamiento de una palabra en un texto es su contexto. El campo de la lingüística que lo estudia es la Semántica distribucional.

> «You shall know a word by the company it keeps» (Firth, J. R. 1957:11)

Los modelos antes citados tienen un problema: dada una palabra, por ejemplo *'banco'*, esta será representada por un vector único independientemente de que sea una palabra polisémica y tenga varios significados. Si bien será un vector que representará bien el uso de *'banco'* dentro del corpus de entrenamiento, será incapaz de distinguir entre varios significados.

Para solucionar esto aparecen los *contextualized word-embeddings* [22]. Estos funcionan siguiendo una idea simple: guardar, además del significado, una referencia al contexto en el que se ha usado. En lugar de analizar un texto palabra por palabra los modelos como ELMo analizan la frase entera.

Los transformers dieron un paso más. Los métodos de *transfer learning* se han usado durante años en el campo de la visión por computador, de hecho han sido las técnicas que han permitido democratizar las soluciones de visión artificial. Estos métodos buscan generar modelos genéricos que luego se pueden especializar en tareas concretas.

Los transformers, y en especial los *Generative Pre-trained Transformers* buscan crear modelos genéricos expertos en un lenguaje que han sido entrenados con corpus muy grandes y pueden servir de primer paso para resolver tareas complejas.

Pasamos de una solución:

$$\textit{Input} \rightarrow \text{Modelo especializado (grande)} \rightarrow \text{Clase}$$

A una solución:

$$\textit{Input} \rightarrow \text{Modelo genérico pre-entrenado} \rightarrow \text{Modelo especializado} \rightarrow \text{Clase}$$

Y finalmente llega BERT o *Deep Bidirectional Transformers for Language Understanding*, que junta conceptos de ELMo y de los transformers generativos. Este

estudio, publicado como solución Open-Source por *Google* supone una revolución en los métodos de procesado PLN.

El uso de ELMo presentaba una clara ventaja sobre los *Generative Pre-trained Transformers* ya que respetaba la bidireccionalidad del contexto, es decir, para entender el contexto de la palabra *'banco'* en la frase:

Ingresa dinero en el banco de la calle Berlín

es quizá más importante su contexto antes de aparición de la palabra que su contexto posterior. *Ingresa dinero* ayuda más a entender el concepto *'banco'* que *de la calle Berlín*.

BERT en cambio usa lo mejor de ambos modelos: la creación de un modelo genérico de los *Generative Pre-trained Transformers* y la bidireccionalidad de ELMo. Esta combinación no se había presentado antes porqué hay un problema importante en el uso de modelos bidireccionales en una solución de modelo multicapa: las palabras son capaces de verse indirectamente a ellas mismas en capas posteriores, haciendo que dejen de funcionar.

> Unfortunately, standard conditional language models can only be trained left-to-right or right-to-left, since bidirectional conditioning would allow each word to indirectly "see itself", and the model could trivially predict the target word in a multi-layered context. [4]

La solución que proponen Jacob Devlin, Ming-Wei Chang, Kenton Lee, y Kristina Toutanova es simple pero efectiva: usar máscaras, demostraron que enmascarando alrededor del 15 % de las palabras del input de manera aleatoria se evitaba el problema.

> In order to train a deep bidirectional representation, we simply mask some percentage of the input tokens at random, and then predict those masked tokens. We refer to this procedure as a "maskedLM" (MLM) [...] In all of our experiments, we mask 15 % of all WordPiece tokens in each sequence at random. [4]

Esta solución, publicada solo hace unos meses, ha cambiado la manera en la que se enfocan los problemas de PLN, consiguiendo resultados mucho mejores que los métodos que se usaban hasta ahora para practicamente todos los tipos de problemas del campo. Es destacable remarcar que en el benchmark de referencia GLUE consiguió una precisión 7,7 % mejor que los demás modelos. [4]

Todos estos avances han ido mejorando y empujando las soluciones existentes para procesar lenguaje natural. Sin embargo vemos también que es un campo relativamente nuevo y de una complejidad enorme y esto estará presente en el desarrollo de todo este proyecto.

3. Análisis

Todo problema relacionado con datos requiere un minuscioso estudio de los mismos. En este caso, y dado que se trata de una implementación para una plataforma en funcionamiento, se presenta también un análisis de los requerimientos tecnológicos y funcionales de la solución.

3.1. Moderación de los mensajes

El problema de la moderación de los mensajes ha ido evolucionando para *Holidog*. Pasando de ser un tema poco relevante durante los inicios de la plataforma a ser un problema clave ahora con presencia en 12 países y con más de 500.000 cuidadores y 300.000 usuarios.

El análisis empieza con los requerimientos de negocio; ¿Cómo se moderan los mensajes actualmente?¿Qué criterios se usan?

3.1.1. Criterios de moderación

Los criterios de moderación de los mensajes han sido decididos para asegurar el correcto funcionamiento de la plataforma. La lista que se entrega a los trabajadores encargados de la moderación es la siguiente:

Ban messages that:

- Indicate a phone number
- Indicate an email
- Indicate a full physical address (street number)
- Indicate a very easy to locate place, monument or any other landmark to meet (usually completed with their car brand/license plate or something else)
- Indicate a third party website on which they can meet (Leboncoin.fr, Facebook), this can be by mentioning their username of the platform or what their profile picture looks like.
- Contains insults of any kind (towards a user or the website)

Documento 1: Lista de requerimientos mensajes

Si bien a priori parecen unos requerimientos sencillos, nos encontramos con las primeras dificultades.

Para un humano, detectar cualquiera de estos puntos es relativamente sencillo. Indiferentemente del país en el que se produzca la comunicación, una persona un poco entrenada es perfectamente capaz de detectar por extrapolación un teléfono, un email o una dirección. Objetivamente es un poco más difícil detectar localizaciones concretas o páginas de terceros, ya que se requiere un poco de conocimiento local. Por ejemplo, un moderador indio puede tener problemas para conocer la topología francesa. Aún así, con un poco de conocimientos del lenguaje, sigue siendo una tarea que un humano puede hacer con mucha precisión basándose solo en el contexto.

Para los métodos de clasificación automática, en cambio, el proceso no es tan simple.

Los tres primeros puntos (teléfonos, emails y direcciones) siguen patrones más o menos estandarizados. Si suponemos que la persona que escribe el mensaje añade un teléfono de manera ordinaria, por ejemplo, usando algo parecido a la notación E-123 [30]:

+(Código de país) - AAA BBB CCC

Detectar este tipo de patrones de manera automática debería ser relativamente sencillo. El mismo tipo de razonamiento se puede extrapolar a las direcciones de correo electrónico y a las direcciones.

Los últimos tres puntos, en cambio, suponen un reto mucho mayor. Para poder saber si un mensaje hace referencia a una localización, el sistema automático debe ser capaz o bien de contrastar si una palabra candidata forma parte de un conjunto de ubicaciones (proceso caro y poco escalable) o bien de ser capaz de entender el mensaje (semántica) y saber que una parte del mensaje habla de una localización (por ejemplo, detectando un complemento circunstancial de lugar).

Nos encontramos con la misma dificultad para detectar insultos y páginas webs o usuarios de redes sociales. Cada país presentará su propia idiosincracia y cambiará la manera de referirse a ellos.

3.1.2. Proceso de moderación

Centremonos ahora en como está *Holidog* solucionando el problema. Actualmente se esta optando por una estrategia manual, hay un equipo de trabajadores remotos que se encargan de validar manualmente cada mensaje que llega a la plataforma. El proceso es el siguiente:

Cuando uno de los actores de la comunicación; los cuidadores (llamados *petsitters* en la plataforma), los usuarios (llamados *clientes*) o los propios trabajadores de *Holidog* (llamados *operadores*) envian un mensaje, este se guarda en un registro de una base de datos relacional con estado pendiente.

Desde un portal dedicado exclusivamente a la moderación de los mensajes, los trabajadores encargados de validarlos pueden ver aquellos mensajes con estado pendiente.

Los moderadores tienen tres acciones disponibles: aceptar el mensaje, cambiando su estado en la base de datos a *accepted*, hacer un pequeño cambio y aceptarlo, en este caso el estado cambia a *amended* y por último prohibir el envío de un mensaje, cambiando el estado a *banned*. Los mensajes con estados *amended* y *accepted* son accesibles entonces por los usuarios a través de la interfaz gráfica de la plataforma y su centro de mensajería.

El equipo de moderadores cuenta ahora con 2 trabajadores por turno (el servicio esta activo 24h los siete días de la semana), llegando a 5/6 por turno en momentos de mucho tráfico. Al ser un servicio de cuidado y custodia para mascotas, en los periodos de vacaciones hay picos considerables de tráfico, y por ende de mensajes a moderar.

Con estos volumenes de tráfico los errores en la moderación, sobretodo los errores en los que se marca como *accepted* un mensaje que debería ser prohibido, cuestan dinero. Desde el equipo de operaciones se hace una auditoría de esta moderación. Esta intenta saber el ratio de errores de los diferentes trabajadores. El resultado de la realizada durante el 2018 es el siguiente:

Mensajes correctamente validados	5.782
Mensajes que deberían ser *amended*	98
Mensajes que deberīan ser *banned*	120
Porcentaje error humano	3.63 %
Dinero potencialmente perdido (15,20€/subscripcion x 3 meses)	9.940,8€

Cuadro 1: Auditoría interna.

3.2. Los datasets y los mensajes

Se han presentado los criterios que hay que seguir para la moderación de los mensajes y cómo se está solucionando el problema en la actualidad. Pasemos ahora a analizar los datasets de mensajes clasificados que la plataforma ha ido guardando durante estos años.

Actualmente *Holidog* trabaja en 12 países, para cada uno hay una instancia diferente de la plataforma. Así pues, disponemos de 12 bases de datos diferentes, cada una con un número de mensajes diferentes.

Dataset	Número de mensajes	Idiomas presentes
Francia	1.910.526	Francés
Reino unido	790.167	Inglés
Italia	500.123	Italiano
España	354.222	Español, Catalán, Gallego
Alemania	147.983	Alemán
Bélgica	142.201	Francés, Flamenco

Cuadro 2: Datasets etiquetados

En países como Bélgica o España en los que hay regiones con idiomas propios, aparecen mensajes en más de un idioma. Estos *datasets* son más complicados de gestionar, pues las técnicas PLN semánticas necesitan saber con seguridad el idioma del mensaje para procesarlo.

A continuación, se plantea un estudio estadístico de los datasets de Francia, Reino Unido y España. Se han tomado estos tres como referencia ya que los modelos de vectorización y procesamiento de texto están mucho más trabajados en idiomas mayoritarios como el inglés, español y francés que en idiomas más minoritarios.

3.2.1. Análisis estadístico de los datasets

El análisis empieza analizando las clases posibles y su distribución. Observando el número de mensajes moderados respecto al total de mensajes.

Francés

	Número de mensajes	**Porcentaje**
Aceptados	1.781.530	93,25 %
banned	76.052	3,98 %
amended	52.944	2,77 %
Total	1.910.526	100 %

Inglés

	Número de mensajes	**Porcentaje**
Aceptados	754.919	95,54 %
banned	23.887	3,02 %
amended	11.361	1,44 %
Total	790.167	100 %

Español

	Número de mensajes	**Porcentaje**
Aceptados	329.896	93,13 %
banned	19.498	5,50 %
amended	4.828	1,37 %
Total	354.222	100 %

Cuadro 3: Distribución de las clases

Los *datasets* presentan un gran desequilibro en la distribución de las clases, este tipo de *datasets* se conocen como *imbalanced datasets*. Es un problema habitual en los problemas de clasificación: una o varias clases tienen una representación mucho menor que las otras, que es mayoritarias. Técnicas como el *oversampling* o *undersampling* nos ayudarán a mitigar este problema. [11] [31]

Vemos que en este caso la mayoría de los mensajes son aceptados por los moderadores, solo un 6,77 %, un 4,46 % y un 6,87 % de los mensajes son o bien editados y enviados (*amended*) o bien directamente prohibidos (*banned*).

Para los modelos automáticos, que aprenden de los datos existentes, esto representa un prejuicio respecto a la clase minoritaria. El modelo tiene muchos ejemplos de la clase predominante (en este caso de mensajes positivos) y muy pocos de la clase minoritaria. En el caso que nos ocupa la clase minoritaria es la más importante: prohibir un mensaje que debería haberse marcado como válido no supone un problema de negocio tan grande como aceptar un mensaje que contiene datos personales que permiten cerrar la operación fuera de la plataforma. Es un problema que hay que solucionar para poder entrenar modelos con un buen rendimiento.

Analicemos ahora la relación entre los tipos de usuarios y la moderación de los mensajes. ¿Hay alguna correlación entre si el usuario es *petsitter* o *cliente* y la moderación resultante?

Francés

	Petsitters	**Clientes**
Aceptados	1.264.554	569.920
banned + amended	31.359	44.693
Total	2,48 %	7,84 %

Inglés

	Petsitters	**Clientes**
Aceptados	542.259	224.021
banned + amended	11.069	12.818
Total	2,04 %	5,72 %

Español

	Petsitters	**Clientes**
Aceptados	234.981	99.743
banned + amended	9.497	10.001
Total	4,04 %	10,02 %

Cuadro 4: Usuarios/Moderación

En la tabla se ve claramente que los clientes tienden más a enviar mensajes que son corregidos o prohibidos. Y tiene sentido, para los *petsitters* la plataforma es un canal de captación de clientes. Si les cancelan el servicio se quedan sin su canal de ventas. Además, el cliente sin subscripción es el que tiene un incentivo económico más grande para saltarse la plataforma (es él el que tiene que pagar la subscripción).

3.2.2. Análisis del contenido de los mensajes

Hasta ahora hemos analizado el contexto de los mensajes, pasamos ahora a analizar el contendido de los mensajes. Para conseguir una idea general de este, usaremos el estudio de las palabras más frecuentes.

Para realizar la representación de este estudio, usaremos *wordclouds* [8]. Esta representación, usada habitualmente en los problemas de PLN, nos permite ver de manera gráfica el contenido de los mensajes. La idea es la siguiente: en la nube de palabras aparecen las palabras más frecuentes de los datasets, cuanto más grande es una palabra, más veces aparece en el dataset, y por consecuencia más importante es.

Dividimos los datasets en el que contiene los mensajes moderados (*amended + banned*) y el que contiene los aceptados. Para cada uno, realizamos un conteo de las palabras de todos los mensajes.

Este procedimiento tiene un problema, si no se preprocesan antes los mensajes, tokens como teléfonos, direcciones o páginas webs estos pasan desapercibidos. Es decir, suponemos que en los mensajes moderados aparecen muchos teléfonos, pero

cada teléfono es diferente realizando un conteo de palabras no veremos los teléfonos como algo relevante, pues la cardinalidad de cada número de teléfono será 1.

Para solucionarlo se tiene que aplicar un primer paso de preprocesamiento. Aplicamos normas RegEX [7] y filtros para realizar las siguientes transfromaciones:

Número de teléfono → phonenum
Número → num
Símbolo de dinero → moneysymb
Sitio web → href
Dirección e-mail → emailref
Símbolos puntuación → Ø

Con estas transformaciones podemos proceder a realizar el recuento de palabras.

Figura 1: Francés: (a) Mensajes aceptados (b) Mensajes baneados

Figura 2: Inglés: (a) Mensajes aceptados (b) Mensajes baneados

Figura 3: Español: (a) Mensajes aceptados (b) Mensajes baneados

Esta representación nos permite ver que efectivamente hay una diferencia clara en el contenido de los mensajes *accepted* y aquellos *banned*. Podemos ver las referencias a lugares: *vivio en*, *cerca de*, *la calle*... y muchas referencias a métodos de contacto fuera de la plataforma: *mi número*, *mi telefono*, números de teléfono transformados (*phonenum*)... en aquellos mensajes que son prohibidos por los moderadores.

En el caso de los mensajes aceptados, podemos ver que el contenido más frecuente es mucho más genérico. Aparecen saludos, referencias a las mascotas y a su cuidado.

Hay que hacer enfásis en que esta representación solo nos ayuda a entender el contenido de manera general. Es decir, al reflejar solo los tokens más usados, tenemos una idea general de aquello que contienen los más de 3 millones de mensajes que contienen los tres datasets.

Pero hay una consideración importante que aparece del análisis individual de los mensajes prohibidos: los usuarios hacen trampas y intentan crear códigos para superar la moderación de los mensajes y enviar contenido prohibido.

Si los usuarios no hicieran trampas, y enviaran solo contenido prohibido por desconocimiento de las normas (en la mayoría de los casos es así), la representación de los mensajes que nos dan las palabras más frecuentes sería suficiente para conocer los datasets. En este caso sin embargo nos encontramos con estos mensajes con códigos de diferentes tipos. Estos no aparecen en números grandes y pocas veces se repite el mismo código. El uso de este tipo de códigos dificulta la detección del contenido prohibido en los mensajes.

Este tipo de códigos complican mucho la solución del problema, tenemos muchos pocos ejemplos de cada caso y estos pierden peso respecto a los demás durante las fases de entrenamiento. Para solucionarlo entran en juego las técnicas de Esteganografía [1], es decir, técnicas que nos ayuden a encontrar códigos y mensajes ocultos dentro de mensajes portadores. Necesitaremos generalizar al máximo la representación de estos mensajes para poder encontrar estos códigos con nuestros modelos.

Analicemos unos cuantos ejemplos de códigos nos permitan entender este tipo de comunicaciones:

Un usuario envia este mensaje:

hola Paula. has cuidado alguna vez algun perro? mi perrita es un american staffordshire de 3 años te lo digo para que lo tengas en cuenta, eso si es muy buena. si quieres hablamos por wasap y concretamos todo ya que por aqui para reservar me cobran mas aparte de lo tuyo 679 06 77 00

Mensaje 2: Ejemplo mensaje banned 1

Este mensaje es marcado como *banned* por los moderadores. El usuario ve que el mensaje ha sido bloqueado y lo reenvía así:

hola Paula. has cuidado alguna vez algun perro? mi perrita es un american staffordshire de 3 años te lo digo para que lo tengas en cuenta, eso si es muy buena. si quieres hablamos por wasap y concretamos todo ya que por aqui para reservar me cobran mas aparte de lo tuyo. mi numero: seis,siete, nueve,cero, seis,siete, siete,cero,cero

Mensaje 3: Ejemplo mensaje banned 2

Se ha cambiado la representación normal del teléfono por una alternativa, para intentar así engañar a los moderadores.

Este tipo de códigos pueden ser considerados fáciles de resolver ya que un mensaje con 9 palabras que representan números (en el dataset español) puede ser interpretado como un mensaje con un teléfono.

Sin embargo aparecen también códigos más complejos como los siguientes(los mensajes han sido descontextualizados y modificados para respetar la privacidad de las partes):

- *Hola,mejor me contacta al 615nueve69111 y hablamos sobre el cuidado,tipo perro,etc.saludos*
- *6Busca 6mi 6mvl!!t0e l7o de7jo x a7qu7í o8k?*
- *fíjSate eOn las mAayuRscuAas RsoloS, es mi fb*
- *Pero (seis) entonces (dos) tienes que pagar (nueve) la cuota para (cinco) que (siete) te pueda (ocho) mostrarte (dos) mis datos (dos) personales (siete). Me avisas entonces para quedar...*

- *Perfecto, aqui te dejo mi numero y contactamos más facilmente vía whatsapp o llamada. Seis ocho dos 99 tres uno 14 (te escribo así el número porque sino no me deja enviarte el mensaje por seguridad)*
- *6*
  62
  44
  57
  41
  (5 mensajes diferentes seguidos)
- *VI l 0 lX VII Vlll l Vll ll jajjajajja xfi entides jajajja*
- *Intento pasartelo escrito de la siguiente forma: estalayi arroba gemail punto com . Ya me dices algo si te llega.*
- *tam ar amar tin ez*
- *S i e t e en C o r t e i n g l e s*
- *Junio, Julio, Enero, Septiembre, Septiembre, Agosto, Febrero, Febrero, Marzo*
- *f b b g d a b i b*

Dada su bajo número de apariciones son muy difíciles de detectar. De hecho, en:

seis siete ocho seis seis dos dos tres ocho

Esta claro que las palabras siguen representando números, en nuestros métodos de tokenización se pueden comparar con otros números y con repeticiones de números.

En cambio un caso como este:

seiXs sietXe ochXo seiXs seiXs doXs doXs treXs ochXo

Se transforma en un caso mucho más complejo de resolver. Sin eliminar la X de los tokens, estas palabras ya no representan números, no se pueden comparar con otros números y su frecuencia en los mensajes es tan baja que las hace pasar desapercibidas (no hay ejemplos suficientes para que sean relevantes).

Si bien existe la opción de listar todos estos métodos, clasificarlos y crear filtros ad-hoc para ellos. No es una solución viable y es interesante analizar como los usuarios hacen trampas para ver porqué no lo es.

El escenario típico es el siguiente:

1. Un usuario envía un mensaje con un método de contacto en él sin usar ningún tipo de código.

2. Un moderados modera el mensaje y notifica al usuario de que el mensaje ha sido *banned*.

3. El usuario prueba otro método más críptico

4. Volvemos al punto 2.

Así, a medida que los usuarios ven que sus códigos son moderados van incrementando su dificultad.

Si creamos filtros manuales para los códigos que existen ahora, los usuarios probarán con nuevos códigos diferentes que no serán filtrados. Los moderadores humanos son capaces de detectar estos nuevos códigos, unos filtros manuales no.

Es un problema importante que tendremos que tener presente en el diseño y análisis de los resultados.

3.3. Requisitos tecnológicos y de integración

Al tratarse de un problema real, con una aplicación real, no podemos perder de vista los requerimientos tecnológicos detrás de la solución.

La plataforma está desarrollada en *.NET*, con bases de datos *SQLServer*. Para cada país dónde *Holidog* opera hay una versión propia de la plataforma. Así pues, la solución tiene que poder integrarse con las diferentes versiones.

Además, debe ser una solución modular, un paso más en su pipeline de tratamiento de mensajes, fácil de mantener y de usar. No puede estar acoplada a los servicios actuales sino que tiene que ser una aplicación independiente y externa.

Para la auditoría del servicio, se necesita crear una interfaz de usuario que permita probar los modelos y ver aquellos modelos que están siendo usados. Debe ser fácil añadir nuevos modelos y cambiar los actuales por otros más nuevos (serialización de los modelos).

La solución tiene que ser buena detectando los mensajes prohibidos, siendo menos importantes los buenos.

Además el coste de mantenimiento y hospedaje de los servicios no puede ser muy alto.

4. Diseño

Con el análisis realizado, podemos preparar el diseño de los experimentos para crear los modelos. Dada la naturaleza del problema, se propondrán varios métodos posibles para cada fase del *pipeline* del mismo.

Entendemos como *pipeline* el conjunto de pasos que hay que realizar para transformar y clasificar un mensaje. Desde el preprocesado del mismo, a su tokenización y a su paso por el modelo predictivo.

4.1. Preprocesado del texto

El primer paso es el preprocesado, antes de su tokenización (conversión a espacio vectorial), podemos hacer una serie de pasos que nos pueden ayudar a su posterior clasificación.

Proponemos cuatro métodos diferentes de preprocesado: Los filtros simples, la eliminación de las palabras vacías, el *stemming* y la *lemmatization* y el reconocimiento de las entidades nombradas.

4.1.1. Filtros RegEX Simples

Como hemos visto en la fase de análisis, existen patrones comunes para dar teléfonos, emails, páginas web y símbolos y referencias a dinero. Los mismos filtros que hemos aplicado para realizar el análisis del contenido de los mensajes pueden aplicarse como un primer paso de preprocesado.

Llamaremos a este paso filtros RegEX simples porqué se limitan a regularizar las apariciones de estos elementos por una representación genérica. Es importante remarcar que la eliminación de símbolos de puntuación es básica para el correcto funcionamiento de la posterior tokenización. *[Ciudad.]* y *[Ciudad]* deberían ser representadas por el mismo token en la fase de tokenización ya que si no se eliminan los símbolos de puntuación serían considerados tokens diferentes. Un ejemplo de este primer paso es el siguiente:

Hola! Tengo 25 años, llámame al número 600400600!

Se transforma en:

Hola Tengo num años llámame al número phonenum

Con este primer paso, conseguimos tener una representación estadísticamente más representativa del contenido del mensaje.

4.1.2. Eliminación de las stop words

Otro concepto importante en el PLN es el de las *stop words* o palabras vacías. [27] En el lenguaje natural usamos palabras que no aportan valor al significado a una frase, son las conocidas como palabras vacías.

En castellano por ejemplo, sabemos que las preposiciones, los pronombres y los artículos no aportan un significado especial que las técnicas de PLN puedan usar.

Podemos quedar cerca de la fuente del parque de la Barceloneta

Aporta el mismo siginificado que:

Podemos quedar cerca fuente parque Barceloneta

Con este paso se pretende eliminar ruido en la tokenización.

Cada idioma tiene su propio conjunto de palabras vacías y no hay una lista universal para todos los problemas. Para cada implementación en los diferentes lenguajes usaremos la lista disponible en el paquete *spacy* [6].

4.1.3. Stemming y Lemmatization

Hemos visto que nos interesa generalizar al máximo los mensajes para facilitar la comparación entre ellos y quedarnos con aquellas palabras que más significado aporten.

Analicemos las cadena de palabras siguientes:

¿Habría alguna posibilidad de cuidar a mis perritos?

¿Hay posibilidades de que cuides a mis perros?

Estos dos mensajes intentan transmitir lo mismo: Preguntar si existe la posibilidad de cuidar a los perros.

Aplicando los dos primeros pasos que hemos planteado estas dos frases se transformarán en dos representaciones diferentes:

Habría posibilidad cuidar perritos

Hay posibilidades cuides perros

Aquí es donde entran el Stemming y la Lemmatization. [12] Son técnicas que nos ayudan a reducir aún más la variabilidad de las posibles palabras a tokenizar, en este caso trabajando sobre las palabras en sí.

El objetivo principal de estas es reducir las formas posibles que puede tomar una palabra y sus derivadas a una forma base común.

Por ejemplo, en el ejemplo anterior, estas buscarían la manera de reducirlas a:

Haber posibilidad cuidar perro

Aunque difieren en la manera de hacerlo; el *stemming* es simplemente un motor de heurísticas que elimina las terminaciones de las palabras, mientras que la *lemmatization* usa procesos de análisis morfológico para hacerlo más correctamente (a cambio de un coste computacional mucho más elevado). Ambas buscan cumplir el mismo objetivo.

En nuestro caso, se usará el motor de *lemmatization* de *spacy* [6] para comprobar y analizar el impacto de usar este tipo de técnicas.

4.1.4. Named entity recognition

La última técnica de preprocesamiento con la que se experimentará es el Reconocimiento de entidades Nombradas (Named entity recognition - NER).

Uno de los problemas comunes que estamos intentando solucionar es el de la comparabilidad de los mensajes. Si dos mensajes son comparables, conseguimos que su tokenización mantenga su comparabilidad, proporcionando mejores resultados.

En nuestro caso de uso tenemos el problema de las entidades y las localizaciones, por ejemplo, los siguientes mensajes presentan este tipo de falta de comparabilidad:

Podemos quedar en el parque de la Ciutadella o en el Mercadona
Podemos quedar en el parque de la Barceloneta o en el Carrefour
o
Envíame un mensaje por WhatsApp
Envíame un mensaje por Facebook

Las dos parejas de mensajes intentan transmitir una localización o un método de contacto a través de una Organización externa. Pero si suponemos que nuestros modelos solo se han entrenado con la primera frase de cada pareja estos serán incapaces de comparar fácilmente Barceloneta con Ciutadella, Carrefour con Mercadona y Facebook con WhatsApp.

Las NER [18] buscan solucionar este problema. Si podemos transformar las frases anteriores a:

Podemos quedar en -localization- o en el -organization-
o
Envíame un mensaje por -organization-

En este ejemplo vemos que los mensajes pasan a tener una comparación 1 a 1. En esto consiste el reconocimento de entidades nombradas: realizar estas sustituciones sobre las entidades.

Para nuestro pipeline vamos a usar de nuevo el paquete *spacy* [6] y su motor de NER.

La primera fase de nuestro *pipeline* es entonces el preprocesado, este constará de estos cuatro posibles pasos. Compararemos en los resultados las mejoras que supone usar estos pasos o no usarlos.

4.2. Tokenización del texto

Analizando el estado del arte del procesamiento del lenguaje natural y la clasificación de texto hemos visto las diferentes técnicas de tokenización que más se usan actualmente.

Dada la novedad de los últimos que se han nombrado: Los *Transformers, BERT o ELMo*, es un poco prematuro plantear un sistema en producción que use este

tipo de métodos dado que su implementación y mantenimiento son mucho más complejas.

Así, se proponen tres tipos diferentes de tokenización:

- Td-idf: nos permitirá tener un baseline bastante bueno para los otros métodos. Además, es el método más rápido de todos los analizados, y para un sistema real-time en producción puede funcionar muy bien.
- word2vec: en este caso no usaremos un modelo *word2vec* pre-entrenado con un corpus genérico sino que buscaremos entrenarlo con nustros datos. Nos da más flexibilidad y nos permitirá que la comparabilidad de los tokens sea coherente dentro de nuestro problema concreto.
- gloVe: para comparar el rendimiento de nuestros *word2vec*, usaremos un modelo gloVe pre-entrenado con corpus genéricos. En este caso usaremos el mismo paquete que hemos presentado en el preprocesado de los mensjaes: *spacy*, que dispone de vectores gloVe pre-entrenados para varios lenguajes.

4.3. Modelos

La selección de los modelos estadísticos que podemos usar es importante para la resolución del problema. Compararemos el resultado en diferentes tipos de modelos:

1. Random Forest: nos servirá como baseline del rendimiento de los modelos.
2. Redes neuronales recurrentes (*Recurrent neural networks - RNN*): durante el análisis de las técnicas de tokenización hemos visto que los problemas de PLN se benefician de la bidireccionalidad. Usaremos dos tipos de redes recurrentes diferentes que permiten tratarla: Las *Long short-term memory* que se usan ampliamente en PLN y las *Gated recurrent units* que siguen los mismos principios pero son un poco más simples (menos parámetros) y rápidas de entrenar. El uso de estas redes recurrentes debería ayudar a la detección de los códigos y patrones.
3. redes neuronales convolucionales: Ampliamente usadas en problemas de visión por computación. Nos servirán de referencia para compararlas con las recurrentes.

4.4. Balanceo de los datasets

Para solucionar el problema del desequilibrio entre las clases, se probarán varias soluciones diferentes:

1. Balancear el *dataset* haciendo un *oversampling* de los mensajes antes del preprocesado sobre el *dataset*. Se hará una muestra del dataset entero, permitiendo la repetición de los mensajes y dando una probabilidad más alta a aquellos

que son etiquetados como *banned*. Con esta técnica se consigue una nueva muestra de los mensajes más balanceada.

2. Balancear el *dataset* haciendo un *oversampling* usando SMOTE [3]. Es una técnica de remuestreo sintético que nos permite obtener nuevas muestras a partir de las existentes. Usa K-NN para generar vectores parecido a sus N vecinos.

3. Balancear el *dataset* durante el entrenamiento del modelo. Otra opción es incrementar el peso de la clase menos representada durante la fase de entrenamiento del modelo. En lugar de hacer un *resampling* de los mensajes en sí, este método trabaja sobre los pesos de cada clase en la función de pérdida. Si se asigna un peso más alto a la clase con menos representación, conseguimos mitigar el desequilibrio del *dataset*.

4.5. Experimentos

Hemos definido hasta ahora las diferentes opciones de todos los elementos posibles de nuestro *pipeline*. Para comprobar su funcionamiento se plantean los siguientes experimentos:

1. RegEX simple + td-idf: la opción más simple de todas, procesar el texto con el filtro regex y aplicar td-idf

2. Preprocesado completo + td-idf: un paso más, buscaremos ver si el preprocesado completo supone un incremento del rendimiento del modelo.

3. RegEX simple + word2vec: entrenamos nuestro propio modelo de word2vec para poder comprobar si usar los *embeddings* propios de nuestro problema supone un incremento de rendimiento.

4. Preprocesado completo + word2vec: buscaremos encontrar si el preprocesado completo supone una mejora usando nuestro modelo word2vec.

5. RegEX simple + GloVe: usaremos los vectores pre-entrenados de *spacy* y nuestro filtro simple.

6. Prerpocesado completo + GloVe: buscaremos encontrar si el preprocesado completo supone una mejora usando el modelo GloVe.

Además, probaremos las diferentes técnicas de rebalanceo para determinar el mejor tamaño y distribución del *dataset*.

Los experimentos se realizarán con el conjunto de mensajes en español. Se trata de uno de tamaño mediano, los métodos que funcionen en el funcionarán incluso mejor en los de tamaño más grande (inglés y francés). La idea detrás de usar este y no uno de los grandes es que el método elegido debe ser capaz de funcionar en todos, los más grandes y los más pequeños. Si lo probáramos sobre un conjunto más grande corremos el riesgo de que los resultados solo funcionen dado su gran cantidad de ejemplos y no sean extrapolables a los de tamaño más pequeño.

5. Resultados

5.1. Determinación del método para balancear los datos

Hemos puesto a prueba los diferentes métodos de rebalanceo para determinar aquel que mejores resultados nos permite conseguir.

Partimos de un conjunto de aproximadamente 350.000 mensajes, de estos, solo unos 25.000 son ejemplos de mensajes *banned*. Estamos hablando de solo un 7,14 %. Si entrenamos un modelo sobre estos datos sin balancear obtenemos lo siguiente:

	Mensajes			**Aceptados**		**Moderados**	
Modelo	Train	Validation	Accuracy	TP	FP	TN	FN
LSTM + td-idf	280k	70k	92,84 %	64.901	17	86	4.996

Cuadro 5: Modelo sobre dataset sin balancear

A primera vista podría parecer un buen resultado, se ha logrado una precisión del 92,84 %, es decir, el modelo esta clasificando bien la gran mayoría de los 70.000 mensajes que usa para validarlo. Pero si nos fijamos en como ha clasificado los mensajes vemos lo siguiente: el modelo está clasificando la mayoría de los mensajes como aceptados, produciendo la siguiente matriz de confusión:

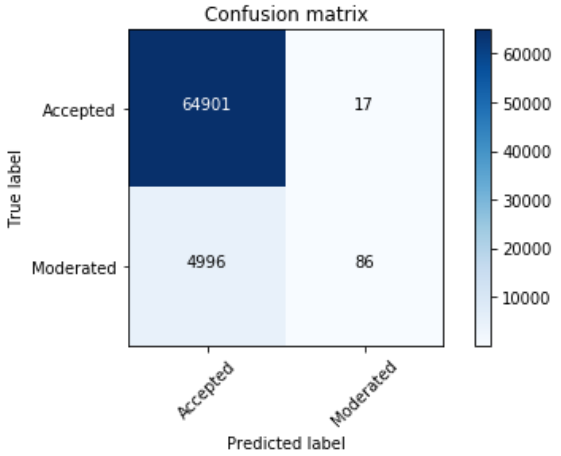


Figura 4: Matriz de confusión sobre el dataset sin balancear

En la matriz vemos que se han clasificado solo 103 mensajes como moderados, y de ellos, solo 86 eran realmente negativos. De los 5.082 mensajes moderados solo se han clasificado correctamente un 1,69 %. La clase minoritaria no consigue ser significativa en el entrenamiento del modelo.

5.1.1. Oversampling con repetición

La manera más simple consiste en hacer un muestreo con repetición de la clase minoritaria. Si queremos conservar una muestra de 350.000 mensajes, y queremos tener una representación equitativa de las dos clases (175.000 mensajes buenos y 175.000 mensajes moderados) seleccionaremos 175.000 mensajes aceptados y repetiremos al azar los 25.000 mensajes hasta conseguir 175.000.

Entrenamos un modelo sobre estos datos y obtenemos unos resultados aparentemente muy buenos. Con 15 *epochs* en un modelo LSTM obtenemos una pérdida muy pequeña (0.3863) y una precisión del 83,78 %.

Aún así, estos resultados no son válidos. Un remuestreo con repetición nos puede ayudar a aumentar el tamaño de la clase minoritaria pero solo nos permite obtener resultados significativos si este aumento es un porcentaje relativamente pequeño. En este caso estamos hablando de un incremento del 500 %, es decir, cada mensaje tendrá alrededor de 5 copias.

Si observamos las curvas de aprendizaje del modelo, podemos comprobar algo extraño:

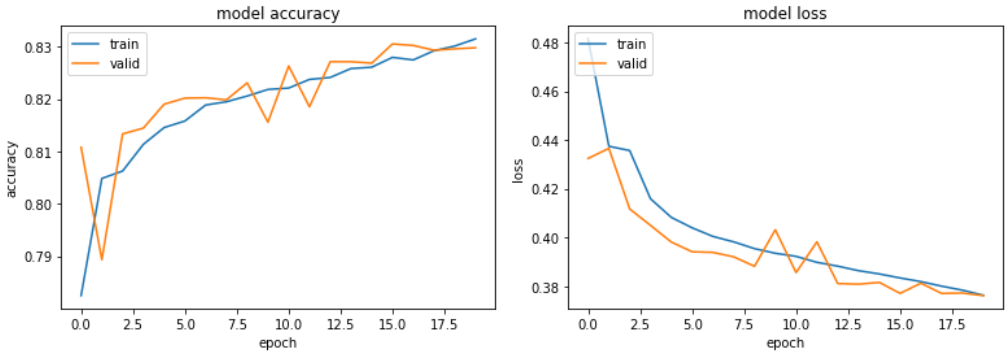


Figura 5: Entrenamiento: (a) Precisión (b) Pérdida

Las curvas de validación son mejores que la de entrenamiento. Este tipo de modelos usa una parte del conjunto de datos de entrenamiento (en este caso un 20 % de los ejemplos) para validar el resultado de cada iteración de entrenamiento. Teniendo en cuenta que en nuestro ejemplo los mensajes moderados aparecen una media de 6 veces, es muy probable de la validación use mensajes que se han usado para el entrenamiento haciendo que las métricas dejen de tener sentido.

5.1.2. Oversampling usando SMOTE

Otra opción es usar SMOTE o *Synthetic minority over-sampling technique* [3], para generar nuevas muestras sintéticas a partir de las que tenemos. En este caso hemos usado los 5 vecinos más próximos de cada muestra de los mensajes moderados para generar vectores de características nuevos.

Conseguimos pasar de un dataset con 25.000 ejemplos negativos a un dataset balanceado 50/50. En este caso se opta por 175.000 mensajes positivos y 175.000 mensajes negativos, con 25.000 originales y 150.000 sintéticos.

En este caso se ha optado por seleccionar 15.000 de los mensajes originales con la proporción original para validar el modelo.

Entrenamos un modelo igual al de los otros ejemplos y obtenemos lo siguiente:

	Mensajes			**Aceptados**		**Moderados**	
Modelo	Train	Validation	Accuracy	TP	FP	TN	FN
LSTM + td-idf	335k	15k (Sin sintéticos)	82,84 %	11.506	2260	920	314

Cuadro 6: Modelo sobre dataset con SMOTE

Con la siguiente matriz de confusión:

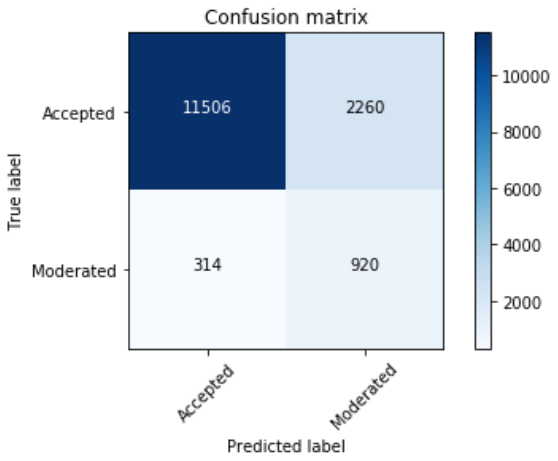


Figura 6: Matriz de confusión del modelo con SMOTE

Con estos resultados vemos que el modelo funciona mucho mejor que el modelo entrenado sobre el dataset sin balancear. Se consigue una precisión más baja (del 82,84 %) pero la clase negativa se predice mucho mejor. Hemos pasado de predecir bien el 1,69 % de los mensajes moderados a el 74 %, consiguiendo así mejorar un resultado totalmente erróneo a uno con el que se puede trabajar.

De todos modos, hay que considerar que el método que hay detrás de SMOTE puede tener problemas al trabajar con lenguaje natural.

Con el método de la repetición, conseguíamos que todos los mensajes, y por ende, sus representaciones en el espacio de características fueran tokenizaciones de texto real, es decir, son mensajes reales con sentido semántico que se traducen en vectores que intentan conservar estos sentidos.

Con SMOTE, el oversampling se hace sobre el espacio de características. Se cogen los vectores que representan a los mensajes y sobre ellos se hacen pequeñas

transformaciones para generar los nuevos ejemplos sintéticos. Estos dejan de ser representantes de texto real, haciendo que los modelos tengan menos ejemplos de vectores con coherencia semántica.

5.1.3. Modificación de los pesos durante el entrenamiento

Podemos usar una estrategia diferente: en lugar de balancear los datos modificándolos, podemos cambiar los pesos de la función de pérdida.

Con modelos basados en pesos, la implementación de este método es sencillo. La idea es que cuando el modelo aprenda de la clase minoritaria los pesos de las funciones se modifiquen con un multiplicador positivo.

Es decir, que cuando se trate con un mensaje moderado, este modifique los pesos como si se tratara de más mensajes positivos. Con esto no conseguimos balancear los datos en sí, pero sí el resultado de la predicción del modelo.

Hagamos la prueba sobre los 350.000 mensajes del dataset español. Sin aplicar esta técnica el resultado ha sido que el modelo era solo capaz de predecir bien el 1,69 % de los mensajes moderados.

Repitamos el mismo experimento cambiando los pesos, para conseguir una representación más equitativa haremos que cada mensaje moderado cuente como 10 mensajes aceptados.

Conseguimos el siguiente resultado:

	Mensajes			**Aceptados**		**Moderados**	
Modelo	Train	Validation	Accuracy	TP	FP	TN	FN
LSTM + td-idf	280k	70k	92,19 %	61.815	3.506	2.720	1.959

Cuadro 7: Modelo sobre dataset con pesos 1/10

Pasamos de un 1,69 % a un 58,13 % solo cambiando los pesos. No es un cambio significativo, con SMOTE hemos conseguido pasar a un 74 %. Sin embargo, este método tiene una ventaja: Se puede combinar perfectamente con los otros métodos.

5.1.4. Undersampling y modificación de pesos

Viendo los resultados obtenidos con los diferentes métodos podemos hacer otra prueba: Reducir el tamaño de los mensajes positivos a 75.000 y mantener los 25.000 negativos, así aumentamos la proporción del 7 % a un 25 %, manteniendo toda la representación de la clase minoritaria.

Hay que tener en cuenta que los mensajes aceptados tienen menos varianza que los mensajes moderados. Así que el impacto de reducir la muestra de estos respecto a los negativos es menor.

A este nuevo dataset modificado le podemos sumar una modificación de los pesos que nos ayude a aumentar la importancia de la clase minoritaria. Si tenemos un 25 % de mensajes de esta clase y hacemos que los pesos de esta clase tengan un factor de x4 obtenemos un resultado mucho más balanceado. Dado el requerimiento de *Holidog* y la mayor importancia de la clase negativa optamos por un factor de x4 en lugar de un factor de x3.

Con este método y entrenando el mismo modelo que en los escenarios anteriores obtenemos:

	Mensajes			**Aceptados**		**Moderados**	
Modelo	Train	Validation	Accuracy	TP	FP	TN	FN
LSTM + td-idf	85k	15k	74,91 %	8.335	3.090	2.902	673

Cuadro 8: Modelo sobre dataset con undersampling y pesos 1/4

Finalmente hemos conseguido un balanceo de los datos que funciona sobre los mensajes moderados. Consiguiendo una precisión del 81,17 % sobre esta clase.

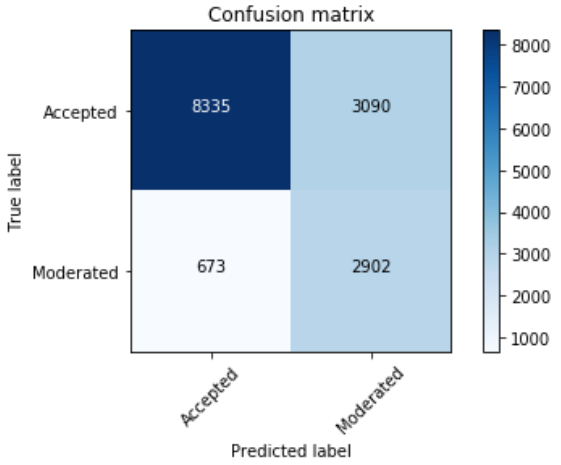


Figura 7: Matriz de confusión del modelo con Undersampling y modificación de los pesos

5.2. Comparación entre los diferentes modelos y métodos de preprocesado

Con el *dataset* correctamente balanceado, usando el último método presentado, pasamos a realizar los diferentes experimentos planteados.

Presentaremos los resultados y analizaremos los resultados obtenidos.

			Aceptados		**Moderados**	
Modelo	Preprocesado	Accuracy	TP	FP	TN	FN
LSTM	Basic + td-idf (128)	70,82 %	7.825	3.600	2.798	741
LSTM	Complete + td-idf (128)	74,91 %	8.335	3.090	2.902	673
CNN	Complete + td-idf (128)	70,96 %	8.398	2.983	2.246	1.373
R. Forest	Complete + td-idf (128)	81,70 %	10.688	667	2.109	1.536
LSTM	**Complete + td-idf (512)**	**81,55 %**	**9.261**	**2.064**	**2.972**	**703**
CNN	Complete + td-idf (512)	49,12 %	4.330	7.092	3.038	540
R. Forest	Complete + td-idf (512)	84,39 %	10.804	572	1.364	2.260
GRU	Basic + Word2Vec	38,97 %	2.762	8.551	3.083	604
R. Forest	Basic + Word2Vec	83,90 %	11.046	258	715	2.981
GRU	Complete + Word2Vec	35,76 %	2.077	9.219	3.287	417
R. Forest	Complete + Word2Vec	80,62 %	11.065	372	1.305	2.258
LSTM	Basic + GloVe	72,59 %	8.037	3.386	2.851	726
R. Forest	Basic + GloVe	82,26 %	10.982	420	1.617	1.981
LSTM	Complete + GloVe	79,87 %	10.404	1.027	1.576	1.993
R. Forest	Complete + GloVe	80,19 %	11.017	430	1.397	2.156

Cuadro 9: Resultados

El experimento con el mejor resultado para nuestro caso de uso es el que se corresponde al modelo LSTM con un preprocesado completo y usando como tokenizador td-idf con un vector de 512 características. Analicemos los resultados para entenderlos.

5.2.1. Tokenizado básico o completo

Tal y como habíamos entendido en la fase de análisis y diseño, vemos que en todos y cada uno de los ejemplos, un preprocesado completo mejora en varios puntos porcentuales los resultados. [3]

Recordemos que usado el método completo, además de las substituciones de teléfonos, webs, emails... usando RegEx también hemos aplicado un proceso de eliminación de las palabras vacías, de *lemmatization* y de extracción de las entidades nombradas.

Es lógica la mejora, pues tal y como habíamos supuesto en la fase de diseño, haciendo estos pasos hacemos que todos los mensajes sean mucho más comparables, facilitando la separación de las clases por los modelos.

5.2.2. Word2Vec

Los ejemplos que han usado nuestro propio modelo preentenado de *word2vec* no han funcionado bien. Es fácil entender el porqué. Para entrenarlo hemos usado

[3] A excepción del ejemplo con word2vec que comentaremos después

nuestro *dataset* balanceado con 100k mensajes.

El sistema *word2vec* entrenado resultante parece válido, las representaciones de las palabras *phonenum* y *whatsapp* están muy juntas en el espacio vectorial. De hecho es capaz de detectar que las palabras más similares a *phonenum* por contexto son:

1. *emailref*
2. *llamar*
3. *whatsap*
4. *escribir*
5. *telf*
6. *whatssap*
7. *tel*
8. *tfno*
9. *tf*

Con este resultado parecería que nuestro modelo tendría que darnos una buena representación en el espacio vectorial, pero no es así.

Como hemos dicho hemos usado 100.000 mensajes para entrenarlo, de estos un 25 % son mensajes moderados y el problema se origina en estos números. 100.000 textos cortos son suficientes para entrenar un modelo especializado para un problema pero no para crear un modelo representativo de la semántica española.

Esto es lo que está pasando en nuestros ejemplos, nuestro *word2vec* es muy bueno detectando mensajes moderados (de hecho en los experimentos en los que se ha usado este método conseguimos los mejores resultados en las clases negativas), pero no es bueno en un contexto más amplio como el de los mensajes aceptados.

Se necesitaría un conjunto de mensajes mucho más grande para poder representar esta semántica completa y poder clasificar mejor los mensajes aceptados.

5.2.3. GloVe

El caso de GloVe también es interesante, los resultados son buenos, consiguiendo resultados parecidos a los mejores, pero vemos que no son muy consistentes. Durante el entrenamiento se observa que la evolución es errática y no sigue una tendencia clara.

El problema en este caso es parecido al anterior. Los vectores GloVe que hemos usado se han entrenado a partir de textos de la *wikipedia*. En concreto *spacy* [6] cuenta con 20.000 vectores GloVe únicos, cada uno de 50 características.

20.000 vectores son más que suficientes para representar la semántica general española, pero no para entender las especificidades de nuestro problema. Si en el caso de nuestro modelo *word2vec* teníamos una tokenización demasiado especifica, en este caso tenemos una demasiado genérica.

5.2.4. Td-idf

Vemos que el método más simple es el que ha obtenido los mejores resultados. El modelo LSTM entrenado con un preprocesado completo y un tokenizado td-idf con un vector de 512 características obtiene una precisión del 81,55 % clasificando además un 80,80 % de los mensajes de la clase minoritaria correctamente.

Nuestro tokenizador basado en *td-idf* proporciona un buen equilibrio entre la especificidad del problema y la necesidad de clasificar bien la clase mayoritaria (los mensajes aceptados).

La idea detrás de su funcionamiento nos permite que aquellas palabras que contribuyen más a la clasificación de los mensajes estén representadas en el vector de características y aunque no tiene en cuenta el contexto de las palabras ni la similitud entre ellas, usando un modelo como LSTM, que trabaja de manera bidireccional, conseguimos un modelo capaz de detectar gran parte de los códigos y mensajes ocultos en los mensajes.

Se han realizado experimentos con vectores de 128 características (las 128 palabras más significativas) y con vectores de 512 características. Es interesante analizar las diferencias:

Como hemos visto, los mensajes a moderar tienen más varianza que aquellos que son aceptados. Los aceptados normalmente tratan de temas más comunes: saludos, precios y experiencia de los cuidadores... mientras que los rechazados que intentan esconder información presentan mucha más variedad entre ellos. Esto tiene un impacto directo usando una tokenización *td-idf*, como hemos visto con este método las palabras menos frecuentes tienden a aportar más al significado del mensaje, así que son seleccionadas para formar parte del vector antes que las mas comunes. Si nuestros mensajes moderados tienen más varianza tienden a tener más palabras raras y obtienen así representación antes que las palabras usadas en los aceptados (las más comunes) en los vectores de características.

Si comparamos los resultados del vector de 128 características con el de 512 características vemos que el de 128 clasifica peor los mensajes aceptados, pero mejor los moderados. Podemos saber así que con 128 tókenes la clase mayoritaria no esta suficientemente representada y nuestro método favorece demasiado a la clase mayoritaria. Con el vector más grande conseguimos mejorar el rendimiento sobre los mensajes aceptados sin empeorar significativamente el de los moderados. Consiguiendo así un incremento de 6,64 puntos en la precisión con un vector que representa correctamente a las dos clases.

5.2.5. Los diferentes modelos estadísticos

Los experimentos se han realizado usando diferentes tipos de modelos para comparar su rendimiento en este problema en concreto.

El mejor resultado se ha obtenido con un modelo LSTM de *deep learning*, analicemos el porqué de cada uno:

Random forest: Lo primero que se nos ocurre al estar delante de este tipo de problemas es si es necesario o no usar técnicas de aprendizaje profundo o si con modelos más sencillos como un *Random Forest* hay suficiente. Por este motivo se decidió comparar todos los resultados de los modelos más complejos con los obtenidos usando este.

Si lo analizamos fríamente nos podríamos decantar por ellos, pues en todos los ejemplos se consigue una precisión varios puntos por encima de los demás. Pero hemos visto que la *accuracy* no lo es todo: Los *Random Forest* son incapaces de solucionar el problema de la Esteganografía (códigos ocultos) y fallan siempre en la clase minoritaria aún usando el mismo dataset balanceado.

Las redes neuronales van un paso más allá, son capaces de aprender los patrones de los códigos y solucionar el problema de la Esteganografía que los *Random Forest* han sido incapaces de resolver. Con una red suficientemente grande seremos capaces de tener perceptrones especializados en encontrar y detectar códigos concretos que los usuarios están usando en los mensajes de entrenamiento. Así que ahora tiene sentido hacernos la siguiente pregunta: ¿Qué tipo de red neuronal necesitamos? ¿Es suficiente con una red neuronal direccional como las *CNN* o necesitamos una red bidireccional más compleja y cara de entrenar?

Redes neuronales convolucionales (CNN): Las redes neuronales convolucionales son rápidas y fáciles de entrenar. Hemos comparado su rendimiento en los experimentos que usan *td-idf* para ver si una arquitectura unidireccional es suficiente para solucionar el problema. Los resultados son claros, para un problema de PLN no son suficientes.

Los resultados obtenidos son interesnates, con el vector de 128 características obtenemos un resultado más o menos correcto, no esta al nivel de los modelos bidireccionales pero obtiene un 70,96 % de precisión. Con el vector de 512 en cambio, los resultados son mucho peores, obtenemos un modelo muy bueno detectando mensajes moderados pero muy malo con los aceptados.

Esto tiene sentido si pensamos en la unidireccionalidad de este tipo de redes, si un perceptron detecta un algún tipo de correlación y le da mucho peso, este peso será arrastrado a las siguientes capas y no se tendrán en cuenta las capas anteriores.

Si analizamos las curvas de aprendizaje vemos que este es muy irregular e inestable y no son suficientemente estables para poderse considerar buenas: El modelo no esta aprendiendo correctamente.

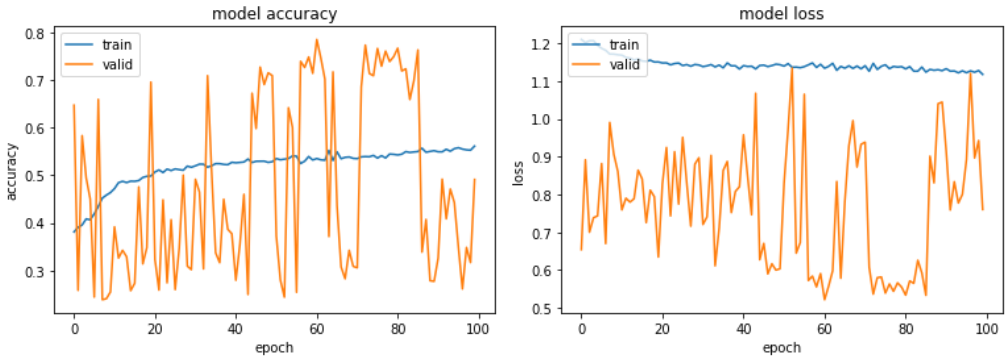


Figura 8: Entrenamiento CNN + td-idf: (a) Precisión (b) Pérdida

Long short-term memory y GRU: Los mejores resultados los hemos obtenido usando modelos de aprendizaje profundo bidireccionales, mucho más caros y difíciles de entrenar. En este problema su uso está justificado: Consiguen un resultado consistente y estable.

La idea detrás de estos modelos es el feedback, las *células* tienen una memoria limitada que les permite trabajar sobre secuencias mucho mejor que los métodos unidireccionales (*feedforward*). El lenguaje natural es una secuencia clara, hemos visto la importancia del contexto y de los códigos en nuestro problema y este tipo de modelos son capaces de aprenderlos y recordarlos. Mejorando así los resultados.

Hemos usado dos tipos de estas redes, en el caso de los ejemplos con word2vec se ha optado por usar una versión simplificada de las LSTM ya que los vectores resultantes eran mucho más complejos y requerían adaptar la capa de inputs a nuestro word2vec entrenado. Para los otros se ha usado la versión completa de las LSTM.

Si analizamos las curvas de aprendizaje vemos que en este caso sí que hay un aprendizaje coherente.

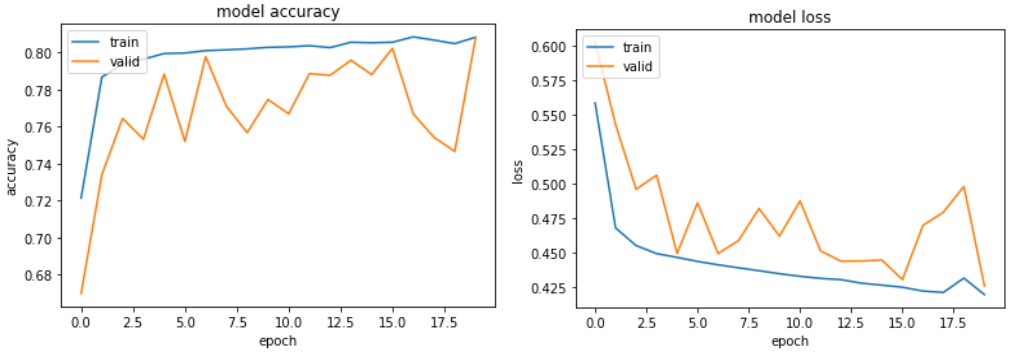


Figura 9: Entrenamiento LSTM : (a) Precisión (b) Pérdida

5.3. Detección de los códigos

Hemos visto la importancia de la detección de los códigos ocultos dentro de los mensajes para solucionar el problema. ¿Cómo se comporta nuestro modelo en estos casos?

Este modelo es capaz de detectar códigos más o menos comunes que los usuarios esconden en los mensajes. Hay margen para mejorar en este campo, que era el objetivo más ambicioso del proyecto, pero por ejemplo el mensaje:

> *hola Paula. has cuidado alguna vez algun perro? mi perrita es un american staffordshire de 3 años te lo digo para que lo tengas en cuenta, eso si es muy buena. si quieres hablamos por wasap y concretamos todo ya que por aqui para reservar me cobran mas aparte de lo tuyo. mi numero: seis,siete, nueve,cero, seis,siete, siete,cero,cero*

Mensaje 4: Ejemplo mensaje banned 2

Presenta un código simple de ocultación del número de teléfono que nuestros métodos de preprocesado no filtran como *phonenum*. En este los dígitos se han sustituido por las palabras que los representan.

Si generamos un mensaje nuevo con este código (para no usar el mensaje original que puede haberse usado para entrenar el modelo):

> *Buenas tardes Antonia, me gustaría que fueras mi cuidadora. seis seis dos dos tres tres ocho nueve seis*

Mensaje 5: Ejemplo mensaje banned 2

El modelo es capaz de detectar el mensaje como negativo con una seguridad del 97 %.

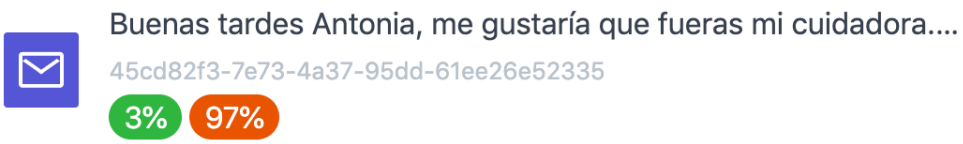


Figura 10: Ejemplo del resultado de la emulación

Se pueden mejorar las técnicas de detección de estos códigos para interceptar también códigos más complejos. Pero tiene poco sentido destinar recursos ahora mismo en hacerlo.

Hemos visto durante la fase de análisis el proceso de generación de estos códigos, que se basa en volver a intentar el envío de un mensaje con un código más complicado una vez el usuario ve que el primero con un código simple ha sido moderado.

Nuestro modelo es muy bueno detectando estos mensajes que deben ser moderados si estos no usan códigos o usan códigos simples. La solución escogida por *Holidog* para evitar que el modelo tenga que tratar con códigos muy complejos es simple:

Si un usuario es detectado haciendo trampas por el modelo, enviando un mensaje que debería ser moderado con un código simple o con un método de contacto a simple vista, el mensaje se marca como *banned* y este usuario es puesto en cuarentena. A partir de entonces las comunicaciones de este usuario serán moderadas manualmente. Si el usuario persiste en intentar enviar mensajes prohibidos con códigos más complejos el equipo de atención al cliente decidirá si debe ser *banneado* de la plataforma.

6. Implementación de la solución

Los modelos generados tienen que poder ser consumidos por las diferentes plataformas de *Holidog*.

Para ello se ha diseñado una API capaz de recibir mensajes de las diferentes versiones de la plataforma (una para cada *dataset*) predecir las probabilidades de ser aceptado o prohibido y devolver al servicio que había solicitado la predicción el resultado de la misma.

6.1. Diseño de la solución

Después de analizar los requerimientos técnicos que nos pedía *Holidog*. El diseño propuesto es el siguiente:

Se creará una API usando *Flask* [20]. Esta recibirá las peticiones de predicción en un endpoint abierto. Estas peticiones serán *POSTs* que contendrán en su cuerpo el mensaje a clasificar y como parámetro el idioma del mensaje.

El número de peticiones a esta API puede ser muy alto en los momentos de más consumo, para poder tratar un número tan elevado de *requests* hay que asegurarnos de dos cosas:

1. El proceso de predicción es la parte más lenta de la API, para no generar un cuello de botella esta tarea se tiene que realizar de manera asíncrona.

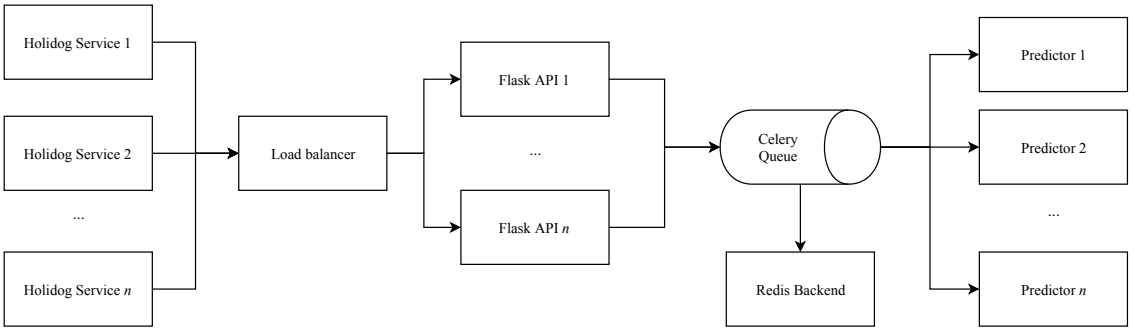
2. La API en sí no debe tener estado. De esta manera se puede escalar de manera horizontal y en momentos de demanda balancear la carga entre varias instancias.

Para solucionar el primer punto se añadirá una cola y un servicio *worker* que consuma las tareas de la cola y los procese (prediciendo su clase). Al usar un framework de python como *Flask*, la solución lógica es usar *Celery* [19].

Celery nos soluciona el problema del asíncronismo, ejerciendo de cola con varios procesos consumidores asociados. Celery te permite escoger dónde se guarda la cola, en este caso que las tareas son simples y necesitamos priorizar el rendimiento por encima de la persistencia: se optará por integrar *Redis* [26] como *backend* de la cola.

Cada predictor usará el modelo correspondiente al idioma que ha recibido como parámetro para poder hacer la predicción.

La arquitectura propuesta hasta ahora es la siguiente:



Con esta conseguimos un sistema con alta disponibilidad y capaz de escalar en todos sus elementos, pero nos falta un punto importante: como gestionamos los modelos.

Necesitamos proponer un método simple de mantener que permita actualizar y entrenar los modelos, además necesitamos que los diferentes *workers* de la cola puedan compartir los mismos modelos aún siendo procesos diferentes.

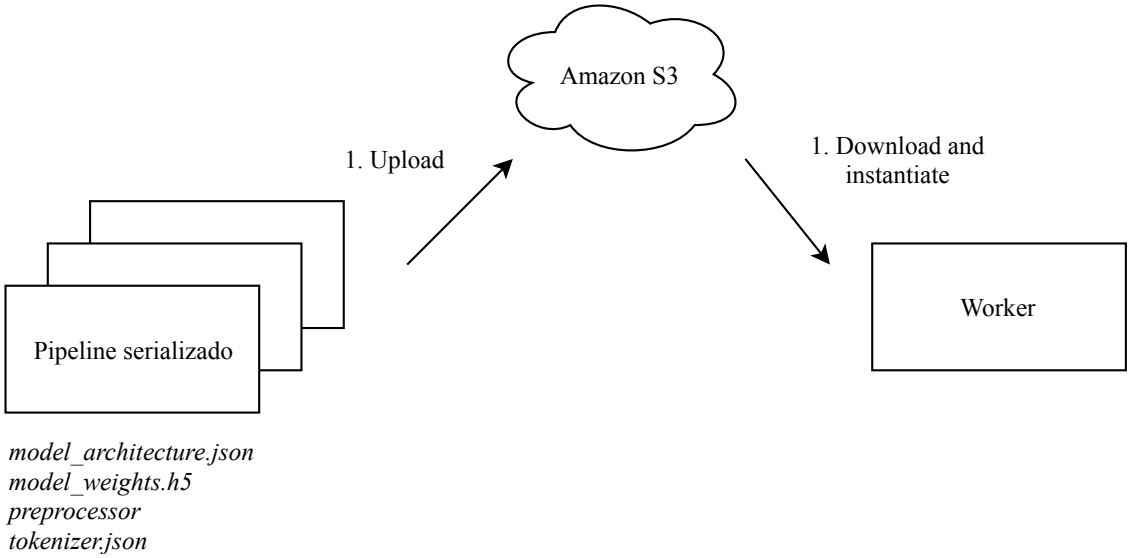
Para solucionar el primer punto la solución propuesta es la siguiente:

1. Los modelos y pipelines entrenados son serializados. Para poder ser cargados en los *workers* y predecir las clases necesitamos serializar y poder recuperar la arquitectura del modelo, los pesos resultantes del entrenamiento, el objeto encargado del preprocesado y el tokenizador.

2. Los objetos resultados de esta serialización son guardados en *S3* [4]. Para cada idioma se crea un directorio y se guardan allí los 4 elementos.

3. Cuando se inicia el servicio de los *workers* estos acceden al contenedor de *S3* y para cada idioma se descargan los artefactos. Con ellos construyen en memoria las instancias de cada paso del *pipeline* que consumirán para predecir las clases de los mensajes.

[4]Servicio de almacenaje de objetos de *Amazon*



Gracias a este proceso conseguimos desacoplar totalmente el uso y el entrenamiento de los modelos, haciendo que los servicios mantengan el principio de especialización.

El último punto a considerar es el uso compartido de los modelos por los *workers*. Estamos hablando de modelos pesados, que tienen que estar siempre disponibles en memoria. Si una máquina tiene más de un *worker* y cada uno necesita poder consultar los modelos de cada idioma si estos no pueden ser compartidos nos encontramos con que el uso de memoria es el siguiente:

*Num. Workers * Num. Modelos * Tamaño de los modelos*

Permitiendo que los modelos sean compartidos conseguimos un uso más racional de los recursos.

Para conseguirlo necesitamos usar *Theano* como librería para nuestros modelos estadísticos. Otras librerías más conocidas como *TensorFlow* no permiten la paralelización de manera sencilla, pues añaden en la memoria del proceso que instancia los modelos el grafo necesario para su ejecución.

Con *Theano* conseguimos que los diferentes procesos *python* puedan acceder al mismo modelo.

Es cierto que esto supone un problema de rendimiento, si dos *workers* quieren predecir usando el modelo del mismo tiempo a la vez, el segundo tendrá que esperarse a que el primero acabe la predicción y libere el recurso. Aún así el considerable ahorro de recursos lo hace una opción lógica para un sistema que trabaja con varios modelos (estamos hablando de procesos que tardan pocos milisegundos).

Además *Holidog* nos requirió la creación de una pequeña webapp para que los moderadores pudieran probar los diferentes modelos y ver como se estaban comportando. Se puede ver la implementación de esta en el apéndice A.

6.2. Deployment de la solución

La solución necesita poder ser consumida por los servicios de *Holidog*, así que necesita desplegarse. Para hacerlo se ha contenerizado usando *Docker* [5] con docker compose. Conseguimos así modularizar cada elemento de la arquitectura propuesta facilitando su despliegue y su mantenimiento.

Los diferentes servicios se han desplegado en un servidor virtual. Exponiendo la API de consulta y el endpoint de la webapp. Como balanceador de carga se ha usado *Traefik*, que hace también de servidor web. Su integración con *Docker* es excelente y nos permite una configuración escalable y sólida.

7. Conclusiones y trabajo futuro

Este proyecto ha presentado una solución viable y escalable para resolver el problema de moderación de mensajes que presentaba la empresa *Holidog*. Los objetivos que se proponían al inicio del proyecto eran los siguientes:

- Investigación de las técnicas de clasificación de texto.
- Investigación de las técnicas del procesamiento del lenguaje natural.
- Analizar y entender los datos de los que la empresa *Holidog* dispone.
- Diseñar un modelo que permita predecir la clasificación de los mensajes y una implementación para poder integrar la solución.
- Implementar el modelo y la solución.

El trabajo empieza analizando cómo la empresa está moderando los mensajes sin ayuda de un proceso automático. Una vez conocido el sistema actual, se ha experimentado con diferentes métodos para tratar de automatizar el proceso.

Los resultados obtenidos con los diferentes experimentos han sido dispares. El modelo diseñado usando LSTM con el tokenizador *td-idf* de 512 características consigue un resultado aceptable para los requerimientos de *Holidog*, con una precisión del 81,55 %. De hecho, la empresa ya lo esta probando y podría usarse en producción. En cambio los modelos diseñados con *GloVe* y *word2vec* no han conseguido resultados tan satisfactorios.

Aunque el sistema propuesto con LSTM obtiene resultados buenos, aún quedan caminos de mejora.

El sistema propuesto es capaz de detectar códigos simples gracias al uso de un modelo bidireccional y a la tokenización usada. Pero el análisis de técnicas más complejas y especializadas de la esteganografía como las que explora Irina Argüelles en su estudio *Lingüística computacional y esteganografía lingüística. Distribuyendo información oculta con recursos mínimos* [17] podrían ayudar a mejorar los resultados.

Otro aspecto que puede ayudar a mejorar el rendimiento de los modelos y conseguir llegar a los niveles de los moderadores humanos es el estudio y aplicación de las nuevas técnicas de PLN como ELMo o BERT. La implementación de estos es costosa y a día de hoy no hay soluciones para su despliegue sencillo en producción.

Además, cabe recordar que los resultados obtenidos con los métodos de tokenizador *word2vec* y *GloVe* no han conseguido el rendimiento necesario. Una solución mixta, usando un método genérico como *GloVe* y uno especializado en nuestro dominio como el *word2vec* que hemos entrenado, podría aportar una mejora significativa. De hecho es algo parecido a lo que propone BERT: especializar un método genérico capaz de detectar y tratar la semántica general de un idioma para luego especializarlo con un método concreto entrenado sobre los datos del problema concreto. Viendo los resultados obtenidos en nuestros experimentos tendría sentido intentar un acercamiento al problema con estas técnicas.

Finalmente, queda también por delante la adaptación de los modelos a los diferentes *datasets*. Por ahora se han creado modelos simplemente extrapolando el español a los otros idiomas. Cada lenguaje tiene unas estructuras semánticas diferentes y la extrapolación directa es peligrosa: idiomas como el inglés, donde la cantidad de estudios y modelos publicados es muy grande, acostumbran a obtener resultados mucho mejores que idiomas más minoritarios. Esto hace necesario un proceso de *fine-tuning* para cada *dataset* si se quieren mejorar las predicciones.

Todos estas mejoras quedan fuera del abasto de este proyecto pero se recomienda su aplicación para mejorar la solución propuesta.

Para concluir podemos decir que hasta ahora el proceso de moderación manual que usaba la empresa Holidog para controlar los mensajes de la plataforma era muy poco escalable. La solución propuesta en este trabajo es capaz de moderar los mensajes de manera automática con un acierto del 81,55 % y escalar a coste constante, ya que es capaz de asumir bien el alto volúmen de mensajes en las temporadas de alta demanda.

Además, esta solución es extrapolable a otros *marketpalces* con problemas similares, teniendo así un evidente interés comercial.

Personalmente, la realización de este proyecto ha sido un reto extremadamente interesante. He aprendido a sobre un campo con el que no estaba muy familiarizado: el procesamiento del lenguaje natural. Es un campo que está en constante crecimiento y que está cambiando la manera en la que entendemos la relación entre las máquinas y las personas.

Apéndice A: Interfaz de usuario para probar los modelos

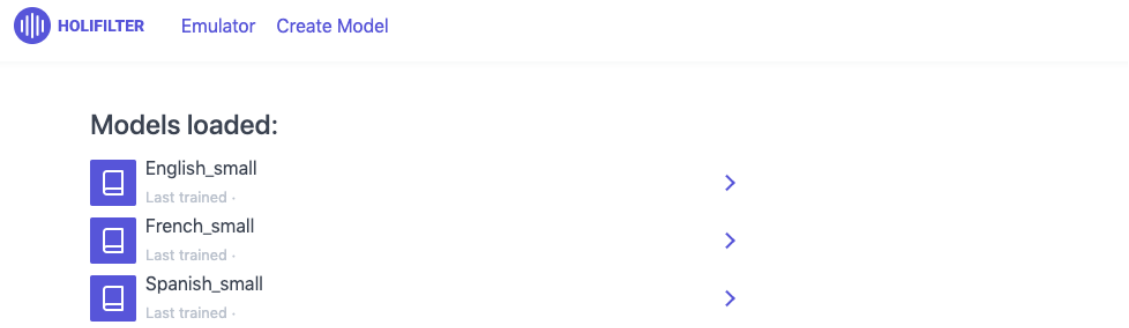


Figura 11: Dashboard 1

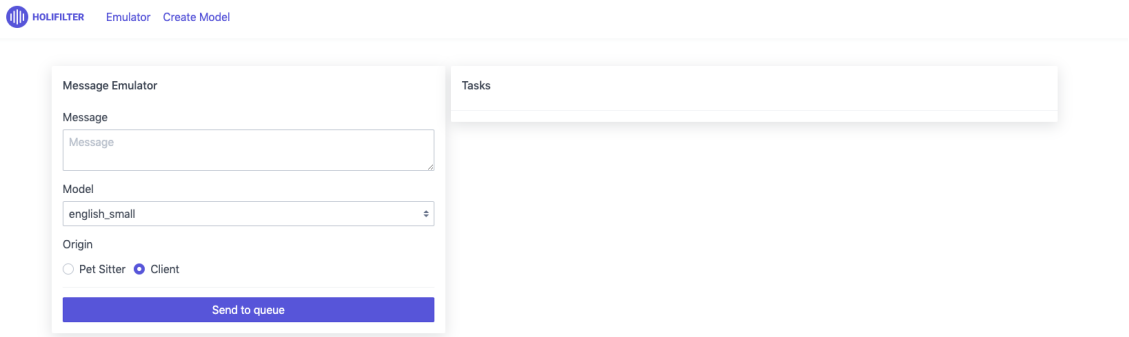


Figura 12: Dashboard 2

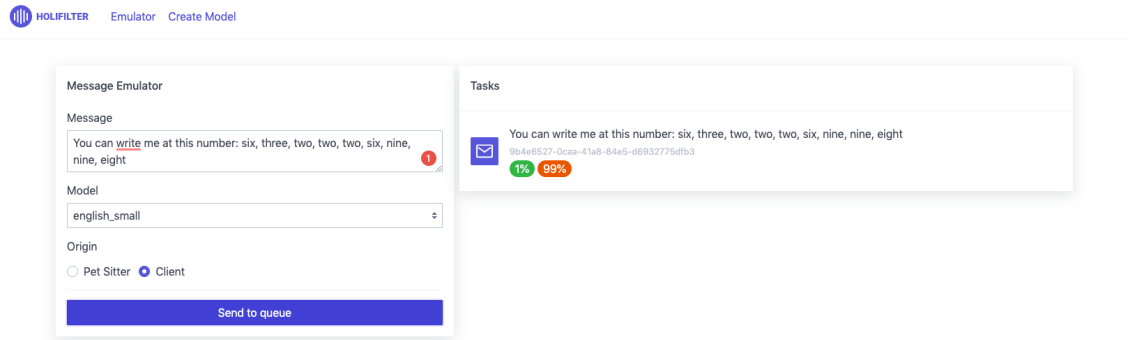


Figura 13: Dashboard 3

Referencias

[1] Carlos Alberto Angulo, Sandra Milena Ocampo, and Luis Hernando Blandon. Una mirada a la esteganografía. *Scientia et technica*, 13(37), 2007.

[2] Leonardo Barón Birchenall, Oliver Müller, et al. La teoría lingüística de noam chomsky: del inicio a la actualidad. *Lenguaje*, 42(2):417–442, 2014.

[3] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[5] Docker, Inc. Docker.

[6] Explosion AI,various. Spacy.

[7] Jan Goyvaerts and Steven Levithan. *Regular expressions cookbook*. O'reilly, 2012.

[8] Florian Heimerl, Steffen Lohmann, Simon Lange, and Thomas Ertl. Word cloud explorer: Text analytics based on word clouds. In *2014 47th Hawaii International Conference on System Sciences*, pages 1833–1842. IEEE, 2014.

[9] John Hutchins. Two precursors of machine translation: Artsrouni and trojanskij. *International Journal of Translation*, 16(1):11–31, 2004.

[10] W John Hutchins. The georgetown-ibm experiment demonstrated in january 1954. In *Conference of the Association for Machine Translation in the Americas*, pages 102–114. Springer, 2004.

[11] Sotiris Kotsiantis, Dimitris Kanellopoulos, Panayiotis Pintelas, et al. Handling imbalanced datasets: A review. *GESTS International Transactions on Computer Science and Engineering*, 30(1):25–36, 2006.

[12] Christopher Manning, Prabhakar Raghavan, and Hinrich Schütze. Introduction to information retrieval. *Natural Language Engineering*, 16(1):100–103, 2010.

[13] Christopher D Manning, Christopher D Manning, and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.

[14] Cade Metz. Google says its ai catches 99,9 percent of gmail spam. *Wired*, 2015.

[15] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[16] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[17] Alfonso Muñoz Muñoz and Irina Argüelles Álvarez. Lingüística computacional y esteganografía lingüística. distribuyendo información oculta con recursos mínimos. *Arbor*, 189(760):021, 2013.

[18] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, 2007.

[19] Open Source. Celery.

[20] Open Source. Flask.

[21] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[22] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[23] Martin Porcheron, Joel E Fischer, Stuart Reeves, and Sarah Sharples. Voice interfaces in everyday life. In *proceedings of the 2018 CHI conference on human factors in computing systems*, page 640. ACM, 2018.

[24] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/languageunsupervised/language understanding paper. pdf*, 2018.

[25] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 133–142. Piscataway, NJ, 2003.

[26] Redis Labs. Redis.

[27] Catarina Silva and Bernardete Ribeiro. The importance of stop word removal on recall values in text categorization. In *Proceedings of the International Joint Conference on Neural Networks, 2003.*, volume 3, pages 1661–1666. IEEE, 2003.

[28] Daniel Stein. Machine translation: Past, present and future. *Language technologies for a multilingual Europe*, 4:5, 2018.

[29] Bruno Trstenjak, Sasa Mikac, and Dzenana Donko. Knn with tf-idf based framework for text categorization. *Procedia Engineering*, 69:1356–1364, 2014.

[30] UIT-T. *Serie E: Explotación general de la red, servicio telefónico, explotación del servicio y factures humanos.*, 02 2001. E.123.

[31] Gang Wu and Edward Y Chang. Class-boundary alignment for imbalanced dataset learning. In *ICML 2003 workshop on learning from imbalanced data sets II, Washington, DC*, pages 49–56, 2003.